2009 Third International Conference on Network and System Security

Policy-Driven Patch Management for Distributed Environments

J. Muhammad, Prof. R.O. Sinnott
National e-Science Centre
University of Glasgow
Glasgow G12 8QQ
jan@dcs.gla.ac.uk

Abstract

e-Science and e-Research is about supporting collaborations especially those that cross administrative boundaries. Typically this is achieved through establishing virtual organizations (VOs) where several institutions and individuals wish to contribute resources for their mutual interest, e.g. to address a given research topic. VOs can be fluid in nature and any individual/cooperating entity may in principle join/leave or have their roles/privileges changed/revoked at any time. Management of such dynamic infrastructures is made more complex since they must address the overall configuration and management of VO-specific resources across multiple sites, as well as configuration and management of the underlying infrastructure upon which the VO exists - referred to in this paper as the fabric. An insecure fabric can undermine the security of collaborating sites and any threat (perceived or real) can often impede the operation of the whole VO. In this paper we present a trust-oriented policy-driven infrastructure that overcomes many of the issues with existing VO models based upon blind trust assumptions of the fabric. Our proposed solution extends the Globus authorization framework involving several decision entities before a patch can be pushed to a target node.

Keywords: Grids, Virtual Organizations, Configuration Management, Patch Management.

1. Introduction

Virtual Organizations (VOs) [1] come in to existence when one or more institutions and their research personnel wish to collaborate and share resources. These resources can be a variety of things including hardware such as storage servers, databases, clusters, supercomputers, or specialized resources such as visualization facilities or telescopes etc [1-3]. VOs are typically realized through the application of Grid technologies and middleware. For large-scale Grids involving multi-site collaborations, the installation and configuration of the software infrastructure needs for particular VOs becomes increasingly demanding and complex for local system administrators at given sites since software targeted to the needs of specific VOs often comes from a variety of sources including remote collaborators. Detailed knowledge of these software tools and their often prototypic nature mean that local administrators are left with a variety of potentially conflicting software and software configuration requirements. This is especially the case when site resources are shared across multiple VOs. Software vendors such as Microsoft and Sophos generally release patches to fix vulnerabilities in their software products, however patching heterogeneous software from multiple providers where dependencies exist between the code bases is much harder to achieve. Patches, if applied correctly can remove vulnerabilities from systems, however in some cases patches may break existing systems if not properly analysed and tested before applying. Furthermore, if applying patches is seen as the '*cure-all*' to software vulnerabilities, then question arises why organisations don't apply them as soon as they are made available? The problem is that patch management is multifaceted, and there are often operational reasons why organisations don't apply them immediately. For example, users with the ability to install new software on their local machines may find that conflicts exist, e.g. with anti-virus or spyware software, which can result in the ineffectiveness of the protection software without the user ever knowing or even seeing a warning message. Other problems with traditional patching systems include software configured for manual update which never actually occurs; users switching off their system earlier than the scheduled time for the update to take place or licenses which may expire causing updates not to happen etc [4].

As a result, many systems are left un-patched for months, even years [5]. According to the Computer Emergency Response Team / Coordination Center (CERT/CC) [6], around 95 percent of security breaches could be prevented by keeping systems up-to-date with appropriate patches [7]. The Slammer virus, which swept the Internet in January 2003, caused network outages all around the world, affecting 911 call centers, airlines, ATMs. However Microsoft released the patch fixing the vulnerability that Slammer exploited six months before the incident. Similarly, Code Red and Nimda wreaked havoc on those companies that were not current with their software patch updates [8].

This lax security process is simply not tenable for many security-oriented VOs. This problem is exacerbated where a single inadequately configured VO resource can endanger all other sites within a given VO if it does not take all appropriate measures to ensure its own configuration and security reflecting

978-0-7695-3838-9/09 $26.00 © 2009 IEEE
DOI 10.1109/NSS.2009.67

both its own specific (autonomous) site policy and the agreements set in place for the VO itself. Whilst advanced authorization infrastructures can protect access to given resources, these security mechanisms will be made redundant if the basic underlying fabric upon which those services exist are themselves inadequately protected. This might be from numerous perspectives: firewalls that have been left open or incorrectly configured; out of date anti-virus software; operating systems or middleware itself that have not been patched with latest updates to name but a few examples. Ensuring that all nodes used across a VO have the necessary operating system, middleware and that the underlying fabrics are properly patched and the most up to date antivirus software protection is something that up until now has been left to the individual VO sites. This is something that cannot be left to chance however. It needs to be managed at the VO level. For example, a VO dealing with medical records or any other highly sensitive data sets demands that resources are protected as far as possible. It is the case that the middleware solutions up to now have been primarily targeted at isolated aspects of security, e.g. defining and enforcing advanced authorization infrastructures which can be used to protect access to particular services or data. We argue that a more integrated solution is needed that deals with end-end VO-wide security including VO-wide security policies and their enforcement which are aligned with local security policies and supporting the security of the underlying fabric upon which that VO is based. This requires an understanding of local and global policies and the ability to securely deliver and configure any software patches or local site configurations. One way in which such update and configuration can be supported is through configuration management tools. However these have to now primarily focused on a single domain and not tackled inter-organisational collaborations. Thus a typical assumption is that the person using the configuration management tool is the local administrator. This is often not the case in inter-organisational collaborations, and remote collaborators may want or need to install patches on a given remote resource if it has been identified as compromised (subject to the remote security policy allowing local administrators to install/patch their local resource).

In this paper we show how VO-wide security policies can be defined, enforced and aligned with local security policies. We show how configuration management tools can be used to configure multiple resources across multiple organizations to improve the overall security of the VO. To illustrate this, we focus on identification of scenarios driving the work forward.

2. Related Work

There are numerous challenges in integrating configuration management tools with VO-specific middleware across multiple sites. Patch management can be thought of a subset of configuration management, as the latter involves configuring firewalls, routers, switches and other appliances used for connectivity within and across organizations. One of the most important to overcome for improved overall security is in how patches are applied to particular VO collaborator nodes.

Huseyin et al [9] point out several reasons for not applying patches. Firstly, there are too many vulnerabilities to patch. [8] states that in an average week, vendors and security organizations typically announce over one hundred and fifty vulnerabilities along with information how to fix them. Secondly, patches cannot always be trusted without testing especially in production environments [10]. Prior to application, each patch should ideally be tested to make sure that it is working properly and does not conflict or have any side effects with existing deployed applications in the system. In some cases, when patching it is necessary to reconfigure the system and/or recode applications so that patches can work without causing any new problems. Thirdly, distribution of patches is often non-standard. Some patches are made available on vendor web sites (e.g. MS WSUS) and automatically pulled when needed. Typically a user is prompted on whether they want to install the downloaded patch, but there is no real guarantee that a given patch will not have any untoward side effects on existing systems. However the vast majority of patches and updates are not automatically updated. Fourthly, after installation many patches require testing and in some cases system restarts for the patch to take effect.

Jung et al. [11] conduct a survey of patch management products and they found that only 4 out of 10 products can provide protection against attacks such as man-in-the middle, forgery and replay. Dominic and Barry [12] propose an automated patch management architecture claiming to be a complete patch management system with important features of initial vulnerability detection, thorough to testing, deployment and reporting and maintenance. This was an ambitious project and is currently a work-in-progress so verification of this system working properly in multiplatform environments has yet to be demonstrated.

In [13], Tian et. al propose an automated vulnerability management using web services and agent-based systems which scan for vulnerabilities in the Operating System version, services and third party applications running. The system sends resource profile information to a web service ‘scanner’ and returns all possible vulnerabilities. Although, this system may work fine in a standalone or static environment, it has not been tested in a multiplatform and heterogeneous environment where several underlying operating systems with wide range of policies may co-exist.

Chang et. al [14] present a patch management model and architectural design for large scale heterogeneous

environments. This architecture is based on five layers, however it does not address environments when there are several collaborators with wide range of site/system policies as is typical in inter-organisational VOs, and where policy conflict detection and resolution are required. Dominic [15] in an attempt to enhance his previous work [12] identifies that a complete framework based on open source community tools is needed to tackle the patch management process in a holistic approach rather than simply patching the holes. Although this effort brings up several ideas for the research community as whole in developing complete patch management solutions, the extension for heterogeneous and multisite collaborative environment is not addressed. Furthermore the proposed architecture has not been rigorously tested in a real environment which makes the author's claims unsubstantiated.

The major configuration management vendors and tools like Microsoft, Altiris, Computer Associates International, IBM Tivoli, LANDesk etc do not have embedded knowledge of patch interrelationships, nor patch analysis capabilities when dealing with heterogeneous software environments comprising multiple software from multiple providers. Thus these tools are not optimized for inter-organisational and heterogeneous patch management, and their use is labor intensive [16].

In summary, applying patches to mitigate against the challenges and dangers of unpatched systems in the face of potential attacks can be a costly business, however the consequences of not applying those updates immediately can be extreme [17].

3. Patch Management support through Configuration Management

Configuration management tools allow for deployment and management and of course configuration of multiple software environments on heterogeneous infrastructures. Such tools have evolved to allow for the installation and configuration of software in a *given* domain and across a closed set of resources, i.e. they do not deal with inter-organisational configuration management as required for VOs. Our previous work [18] undertook an evaluation of three leading configuration management tool sets: Cfengine [19], SmartFrog [20] and OGSAconfig [21]. We showed how it is possible to use these tools to deploy a variety of Grid software, other software (such as Java) and indeed support data management activities that are commonly required for a particular VO. The work described here extends that work through dealing with the security aspects of exploitation of configuration management software tools for VO-specific patch management.

VO security can be achieved in two principle ways: decentralized security where trust relationships exist between different organizations which are used to define and enforce local access control decisions, e.g. bilateral agreements. Alternatively, centralised security models assume some agreement across all of the VO-sites and users that are participating in a given collaboration. One example of the former is the Internet2 Shibboleth technology where multiple individual federated Identity Providers (IdP) can be used for authentication and delivery of signed attributes to service providers (SP) which can subsequently be used for access control. An example of centralized security is the Virtual Organisation Membership Service (VOMS) where a centralized attribute authority is defined and used to provide credentials used for access control decisions. Hybrid combinations of these centralized and decentralized security models are possible. The advantages and disadvantages of these models are compared in [22].

The actual authorization decisions that are made can be undertaken in a variety of ways using a variety of technologies: Role Based Access Control (RBAC) is one of the more common approaches. The PERMIS technology (www.permis.org) provides an X509 based RBAC infrastructure that has been extended to work with a variety of middleware commonly used to support VOs including the Globus Grid middleware (www.globus.org) and the Open Middleware Infrastructure Initiative (OMII-UK – www.omii.ac.uk) and the Internet2 Shibboleth technology. Through digitally signed Security Assertion Markup Language (SAML) [23] assertions, finer grained security can be supported where VO-specific roles can be delivered and subsequently used to determine for example whether the credentials that are supplied (either through Shibboleth or from VOMS) meet the security requirements from the local (protected) service provider.

To explore the challenges of VO-specific patch management, we consider both centralized and decentralized models for patch management in a distributed environment. In particular we focus upon VO-specific patch management across multiple collaborating sites. For brevity we assume use of CFengine as the configuration management tool but this can be generalized to other tools such as SmartFrog or OGSAconfig.

CFengine [19] is an open source tool widely used for policy host configuration systems. It includes a high level policy language and a low level logic engine. At start-up, CFengine retrieves a policy definition from a central place (server) and then tries to adapt the local system to that policy. The *cfenvd* daemon which is part of CFengine collects local system statistics in order to detect abnormal behaviour [2].

In [18] we assumed a single domain model where the privileges required to run this script existed across all nodes where deployment and configuration was required. However in inter-organisational collaborations such assumptions are rarely likely to be the case. Rather precise specification of the rules and regulations on who is allowed to install what patch and

configure what software on which resources needs precise specification and subsequent enforcement. Furthermore understanding of the software dependencies on the deployed nodes and the potential vulnerabilities that may or may not exist on a given node must be clearly understood before any patch management can be achieved.

4. Proposed Solution

To tackle this, two key components of the infrastructure that we put forward for security-oriented patching of particular VOs through configuration management include:

- **VO-Knowledge Base** (VO_{kb}): in our current implementation this is a MySQL database that contains an inventory of all information of the nodes in the VO. This includes the specific operating system version, the service packs/RPMs, patch information, applications, libraries and antivirus softwares installed on the VO nodes.
- **VO-Vulnerability Data Base** (VO_{vb}): contains announced vulnerabilities and advisories from recognized sources of information for that particular VO. This might be through information sources such as CVE [24], Bugtraq [25], CERT [6], or recognized middleware providers such as Globus etc.

To understand the interplay of these components in the patching and configuration of a particular VO we outline typical interactions that take place as shown in Figure 1. To being with we assume that a VO administrator (VO_{ad}) receives an alert from a recognized software authority that a given vulnerability has been found in a particular software component. The VO administrator retrieves this software update and checks that it has been digitally signed by the recognized authority, e.g. that the MD5 checksums are correct, and when this is the case they check with the VO_{kb} as to whether any of the nodes existing in the VO are affected. When this is the case, information on the update is added to the VO_{vb}. This includes meta-data on the update such as where it was downloaded from; who validated its authenticity; the software vulnerability it addresses and the nodes that are impacted by this update.

We note access to the VO_{kb} and VO_{vb} is restricted and requires fine grained authorization. This is achieved through VO_{ad} specific credentials being required when querying or updating the VO_{kb} or VO_{vb}. We also note that it is quite possible for multiple individuals to hold the VO_{ad} role or indeed have different roles that are allowed access to subsets of the information in VO_{kb}/VO_{vb}. This might be the case for example when an administrator can only administer subsets of the resources across the VO, e.g. only those in a given institution for example. It is equally possible that a VO_{ad} can delegate their credentials to other known and trusted individuals subject to VO policy. One way that this can be achieved is through the JISC funded DyVOSE project Delegating Issuing Service (DIS) which is described in [26].

The default for access to VO-specific resources is set to deny, i.e. unless authentic and valid credentials are presented then access to these resources is denied. Once authorized access to the VO_{kb} and VO_{vb} has shown that there are indeed nodes across the VO that are impacted by this particular update, i.e. a security hole exists, a CFengine script is auto-generated and parameterized with the update itself; the hosts that are affected and the credentials of the VO administrator. An execution plan for the CFengine script is subsequently derived and sent to each node that is affected along with the credentials of the VO_{ad}. Each node that receives this update will use the VO_{ad} credentials and the update information itself to make a local authorisation decision. We emphasise this point since it is at the heart of the solution we propose. A given resource provider must be able to guarantee that a request to install an update or configure a given local resources comes from an authentic, valid and above all trusted authority.

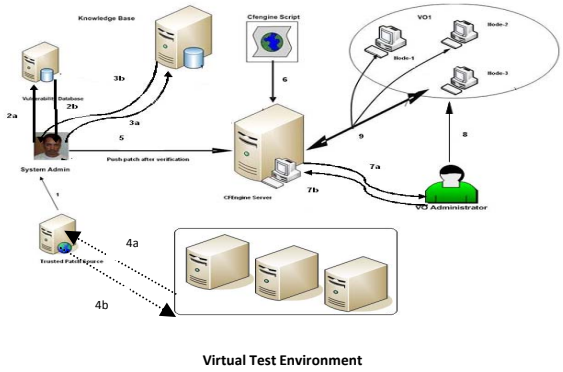

Figure 1: Policy-driven patch management in a single VO environment

In Figure 1, to make sure that the target patch does not cause any untoward effects on nodes it needs to be tested for conflicts/dependencies issues. This is achieved by testing it in a virtual test environment. Once all these tests have been passed by the system administrator can push the received patch to the central CFengine server. At this point an authorization decision is made according to central policy specified to ensure that the system-administrator is allowed to push this patch through to the CFengine server. Based upon the privileges of the system administrator, a decision is made as to which nodes the administrator's patch is allowed to be sent to. This decision will be based both on the VO policy to which the node belongs to and its trust relationship with the CFengine server; the patch itself and the parameters such as authorization credentials for remote administrators, the target node, agreed with VO policy. If privileges allow then the CFengine will submit the CFengine script to the remote nodes where it will be intercepted by a local policy enforcement point (PEP) which restricts access

to a local service which itself protects access to a local CFengine daemon. However, the 'allow' decision will be based on the fact that if node is affected; there are no conflicts, and the VO policy is satisfied. Assuming that the privileges presented are authentic, sufficient and agree with the local policy and there are no dependencies or side-effects on the target node, the script will be forwarded to the daemon and installation and configuration of the patch can takes place. The workflow of the above mentioned activities are demonstrated below by showing Authorization decisions in Figure 2.

At layer-1 (patch arrival & analysis layer) a system admin receives a patch from a software vendor (step-1). At (step 2) the received patch will be further confirmed for its legitimacy from independent sources such as Bugtraq etc at the VO vulnerability database (VODB). If no vendor digital signatures are found the patch will be dropped straight away (step 3); in case of a genuine patch containing authentic digital signatures it would be forwarded to the VO_{kb} to check if any affected target nodes exist in the VO (step 4). In case no node is affected the patch is dropped (step 5) otherwise if an affected node is found a positive decision will come (step 6)

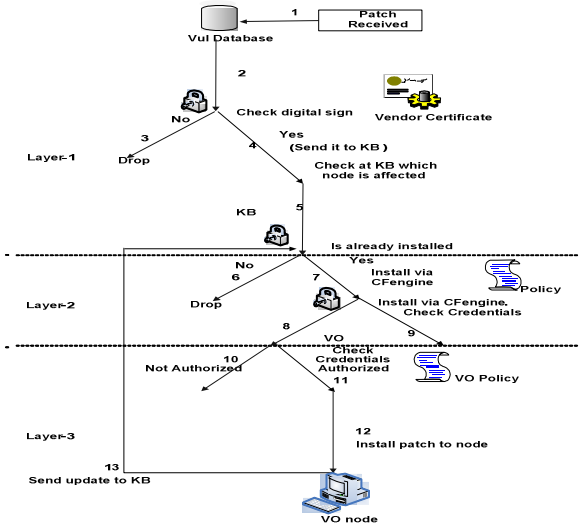

Figure 2: Workflow of the patch activities from receiving to installation

At layer-2 (patch test layer), the received patch is tested in a virtual environment containing an image of the target node. At this layer once the patch has been successfully tested (step 7) it would be sent to the central Cfengine server; in case of test failure (step 8) the patch would be dropped. At this level the authorization decisions will depend upon the presented credentials of the VO administrator (VO_{ad}). If the presented credentials are not valid or have been revoked then a 'deny' decision will be issued (step 9). Otherwise, if the credentials are authentic and valid an allow decision will be made and the script and other parameters such as target nodes for installation via *CFservd* 'server daemon' will be sent to the affected nodes. We note that at the central Cfengine server the *cfservd* is protected by the central policy which requires that only administrators with valid credentials can install patches on the VO nodes. If the system admins credentials have been revoked or are not valid then access will be denied (step-10).

At layer-3 (installation layer), the local VO policy is used to check the credentials for the installation of patch to take place i.e. sites are autonomous. If a system admin has valid credentials in accordance with the local policy then the patch is accepted (step-12) else access is denied (step-11). Once an authorization decision is made in favor of the remote administrator (step 13) the patch installation can take place and feedback/update about successful installation of the patch is reported back to the knowledge base (step-14).

5. Authorization Components & Implementation

To address the security requirements of Figure 3, fine grained access control is needed. Contemporary role based access control models (RBAC) such as PERMIS [27] are not sufficient in detecting conflicting situations/scenarios which involve more than one subject or more precisely PERMIS only supports local authorization decisions where in reality VO-wide authorization decisions are needed, e.g. combining decisions from multiple PDPs simultaneously. Extended RBAC models such as eXtended Access Control Markup Language (XACML) [28] support such situations when there are several entities involved in the decision making process. Our implementation accommodates credentials/attributes certificates issued by several entities and policies expressed by dissimilar domains, e.g. using Access Control Lists (ACL), Security Assertion Markup Language (SAML) callouts, centralized issued Virtual Organization Membership service (VOMS) [29] attributes or decentralized Shibboleth issued credentials. The services they protect are implemented in Globus.

5.1 The GT4 Authorization Framework

The Web Services Resource Framework (WSRF) offers a convergence of Grid and Web services. WSRF offers specifications for interoperability of Grid security. In a particular Grid system, each domain has its own security policy such as a grid-mapfile, ACL, CAS, SAML authorization decision and/or XACML policy statements. Since not all of the above mentioned authorization requirements were addressed in the XACML specification. Lang et. al [30] designed and implemented the Globus Toolkit version4 (GT4) authorization framework. The GT4 authorization framework implements SAML and uses the XACML model. It is composed of one or more Policy Enforcement Points (PEP), Policy Decision Points (PDP), and Policy Information Points (PIP). We have

extended the implementation of the Globus Authorization (GT) framework. When a request for a patch update arrives at the Globus protected CFengine server, the PEP in the Globus authorization engine intercepts it and sends a decision request to the Globus PDP which in turn calls a Master PDP (extended from Globus authorization framework). The master PDP collects information needed by calling multiple PDPs as specified in the security configuration file. These include PDPs for the VO_{KB}, VO_{vb} and the global VO-policy. When the master PDP receives the decisions returned by each PDP, it combines the decisions, using a policy combination algorithm to render a final decision, and returns the decision to the PEP. The PEP then executes the decision, either denying or permitting the request. The master PDP which is responsible for decision making on who is authorized to access/invoke *cfservd* daemon has been designed to be extensible. Currently these checks whether the patch is in the vulnerability list in the database (PDP_{vo}) whether the patch effects any VO nodes and also to make sure that any request/actions are in accordance with the overall VO policy and that no violation VO rules exists. If any of these PDPs return a deny then the patch is not installed and a deny exception is raised. Numerous refinements to this policy are possible. Thus it might be the case that a subset of nodes only needs to be patched. Once an authorization decision is reached by the Master PDP it may still be the case that a given local PDP denies access. This might be for example if that node is currently running jobs for example and patching at that time is not possible. We are currently looking at extensions to the authorization framework to support such scenarios.

6. Conclusions and Future Work

In this paper we have presented a policy based approach for installing and dealing with patches across VO nodes exploiting configuration management tools such as CFengine. We have shown how it is possible to dynamically express policies. A key aspect of our work is how we can use a variety of security mechanisms to install and configure patches. We have exploited CFengine and extend GT4 authorization framework for this purpose.

The work has identified numerous open issues with the patch management of distributed systems like. The work described here is still quite nascent and we expect to explore a range of future research directions. Examples of the kinds of research challenges we expect to explore include issues of dependency management, e.g. when conflicting software requirements exist. How do remove those conflicts if software is already installed across nodes in a VO? Other aspects we are considering using virtualization techniques to test patches before they are pushed to VO nodes. Similarly the work of cloud computing has direct parallels with what we propose here. The work of inter-disciplinary/inter-cloud interoperability can benefit greatly from the work described here.

In the future, an extension to this work will be looking the pros and cons of centralized and decentralized models for security policies. Key to adoption of this kind of approach are standards and technologies that facilitate the expression and enforcement of patch management policies. One issue we have already identified in this is the level of granularity in expressing patch installation and rollback privileges. That is, we wish to allow VO administrators to install software pertinent to their VO, but not allow unrestricted access to install arbitrary code.

7. References

1. Foster, I. and C. KESSELMAN, *The Grid: Blueprint for a New Computing Infrastructure*. 2nd ed. 1999: Morgan Kaufmann.
2. Berman, F., G. Fox, and A.J.G. Hey, *Grid Computing: Making the Global Infrastructure a Reality*. 2003, New York, NY, USA: John Wiley & Sons, Inc.
3. Parashar, M. and C.A. Lee. *Special Issue on Grid Computing*. 2005.
4. Prince, K., *Malicious Software Defense: Have we moved beyond the need for anti-virus and spyware protection software?* 2007, Perimeter eSecurity.
5. Shostack, A. (2003) *Quantifying Patch Management*. **Volume**,
6. *Computer Emergency Response Team/Coordination Center (CERT/CC)*. [cited; Available from: http://www.cert.org/.
7. Dacey, R.F., *Effective patch management is critical to mitigating software vulnerabilities*, in *General Accounting Office*. 2003, United States General Accounting Office.
8. Lynda, M., *Software Patch Management—The New Frontier*, in *Secure Business Quarterly* 2003.
9. Cavusoglu, H., H. Cavusoglu, and J. Zhang. *ECONOMICS OF SECURITY PATCH MANAGEMENT*. in *The Fifth Workshop on the Economics of Information Security (WEIS 2006)*. 2006. Robinson College, University of Cambridge, England.
10. Donner, M. (2003) *Patch Management -- Bits, Bad Guys, and Bucks!* Secure Business Quarterly **Volume**,
11. Seo, J.-T., et al., *Design and Implementation of a Patch Management System to Remove Security Vulnerability in Multi-platforms* in *Fuzzy Systems and Knowledge Discovery*. 2006, Springer Berlin / Heidelberg. p. 716-724.
12. White, D. and B. Irwin. *A Unified Patch Management Architecture*. in *Southern African Telecommunication Networks and Application Conference* 2004.
13. Tian, H.T., et al., *Grid and Cooperative Computing*. Lecture Notes in Computer Science. Vol. 3032/2004. 2004: Springer Berlin / Heidelberg. 1067-1070.
14. Chang, C.-W., et al. *A cross-site patch management model and architecture design for large scale heterogeneous environment*. in *39th Annual International Carnahan Conference on Security Technology, 2005. CCST* 2005: IEEE.
15. White, D. *A UNIFIED ARCHITECTURE FOR AUTOMATIC SOFTWARE UPDATES*. 2004. Information Security South Africa (InfoSec).
16. Colville, R. and M. Nicolett, *Patch Management: Identifying the Vendor Landscape*. Gartner, 2003.
17. Anonymous (2002) *Overview of Attack Trends*. **Volume**,
18. Sinnott, R.O., J. Muhammad, and W. Yuxiang. *Deployment of Grids through Integrated Configuration Management*. in *Parallel and Distributed Computing and Networks (PDCN)* 2008. Innsbruck, Austria.
19. *http://www.cfengine.org/*.
20. *SmartFrog*. *http://sourceforge.net/projects/smartfrog*.
21. *http://groups.inf.ed.ac.uk/ogsaconfig*.
22. Sinnott, R.O. et al. *Advanced Security for Virtual Organizations: The Pros and Cons of Centralized vs Decentralized Security Models*. in *8th IEEE International Symposium on Cluster Computing and the Grid*. 2008. Lyon, France: IEEE Computer Society.
23. Anderson, A., *SAML 2.0 profile of XACML*. 2004, Sun Microsystems.
24. *Common Vulnerabilities and Exposures (CVE)*. http://cve.mitre.org/.
25. *Bugtraq*. http://www.securityfocus.com/.
26. *DyVOSE project*. www.nesc.ac.uk/hub/projects/dyvose.
27. *PrivilEge and Role Management Infrastructure Standards Validation (PERMIS)*. http://www.permis.org/en/index.html.
28. Moses, T., *XACML profile for Web-services*. 2003, OASIS.
29. *Virtual Organization Management Service (VOMS)*.
30. Lang, B., et al., *A Multipolicy Authorization Framework for Grid Security*. 2006, Argonne National Laboratory